

# VIDEO BASIC

20 VIDEOLEZIONI DI BASIC  
PER IMPARARE CON LO SPECTRUM



**GRUPPO  
EDITORIALE  
JACKSON**

*CPU: organizzazione  
esterna e interna  
I BUS*

*Il linguaggio macchina  
Vantaggi e svantaggi  
dell'Assembler  
USR*

*Memorizzazione di programmi  
in linguaggio macchina  
Videosercizi*

*Videogioco n° 18*

# 18

# Spectrum

16K/48K/PLUS



## VIDEO BASIC SPECTRUM

Pubblicazione quattordicinale  
edita dal Gruppo Editoriale Jackson

### Direttore Responsabile:

Giampietro Zanga

### Direttore e Coordinatore

Editoriale: Roberto Pancaldi

Autore: Softidea - Via Indipendenza 88 - Como

### Redazione software:

Francesco Franceschini, Roberto Rossi,

Alberto Parodi, Luca Valnegri

### Segretaria di Redazione:

Marta Menegardo

### Progetto grafico:

Studio Nuovaidea - Via Longhi 16 - Milano

### Impaginazione:

Silvana Corbelli

### Illustrazioni:

Cinzia Ferrari, Silvano Scolari

### Fotografie:

Marcello Longhini

### Distribuzione: SODIP

Via Zuretti, 12 - Milano

### Fotocomposizione: Lineacomp S.r.l.

Via Rosellini, 12 - Milano

### Stampa: Grafika '78

Via Trieste, 20 - Pioltello (MI)

### Direzione e Redazione:

Via Rosellini, 12 - 20124 Milano

Tel. 02/6880951/5

Tutti i diritti di riproduzione e pubblicazione di  
disegni, fotografie, testi sono riservati.

© Gruppo Editoriale Jackson 1985.

Autorizzazione alla pubblicazione Tribunale di  
Milano n° 422 del 22-9-1984

Spedizione in abbonamento postale Gruppo II/70  
(autorizzazione della Direzione Provinciale delle  
PPTT di Milano).

Prezzo del fascicolo L. 8.000

Abbonamento comprensivo di 5 raccoglitori L. 165.000

I versamenti vanno indirizzati a: Gruppo

Editoriale Jackson S.r.l. - Via Rosellini, 12

20124 Milano, mediante emissione di assegno

bancario o cartolina vaglia oppure

utilizzando il c.c.p. n° 11666203.

I numeri arretrati possono essere

richiesti direttamente all'editore

inviando L. 10.000 cdu. mediante assegno

bancario o vaglia postale o francobolli.

Non vengono effettuate spedizioni contrassegno.



**Gruppo Editoriale  
Jackson**

# SOMMARIO

## HARDWARE ..... 2

La CPU. Organizzazione esterna  
di una CPU. I bus.

## IL LINGUAGGIO ..... 10

Il linguaggio macchina. Svantaggi  
e vantaggi dell'Assembler.  
USR

## LA PROGRAMMAZIONE ..... 22

Alla scoperta dell'Assembler.  
La numerazione esadecimale.  
Come memorizzare i programmi in  
linguaggio macchina. Esempi  
Assembler.  
Conversione di un carattere in LM.

## VIDEOESERCIZI ..... 32

## Introduzione

*Prima di andare avanti ... facciamo un  
passo indietro. Per introdurre, infatti, il  
tanto famigerato linguaggio macchina  
(l. m. per chi vuol darsi un tono) e  
l'altrettanto noto assembler, occorre  
rinfrescare i concetti base nella CPU,  
la sua organizzazione interna e  
esterna, i bus.*

*Non è un'inutile esercitazione  
accademica; per ottenere mirabolanti  
prestazioni dal proprio computer è  
necessario - a volte indispensabile  
quando si vuole risparmiare  
occupazione di memoria e tempo di  
esecuzione - programmare  
direttamente il microprocessore che lo  
governa.*

# HARDWARE

## La CPU

Il termine CPU è spesso usato con due distinti significati. Considerando il computer completo di periferiche, si indica talvolta con CPU la parte contenente l'unità centrale propriamente detta (quella che esegue le istruzioni nella corretta sequenza), la memoria centrale e le interfacce di collegamento con il resto del calcolatore (e, in molti personal, anche la tastiera). In questo caso CPU (o unità centrale) è sinonimo di: "il computer vero e proprio, escluse le periferiche fisicamente separate (video, registratore, stampante, ecc.)".

Dal punto di vista logico, invece, la CPU è solamente l'unità centrale di calcolo e di controllo, escludendo quindi memorie ed interfacce di qualsiasi tipo. Visto che noi abbiamo sempre inteso il termine "CPU" (ed anche quello di "unità centrale") solo in questo secondo significato, continueremo a farlo anche nel resto dei nostri discorsi.

Fatta questa doverosa precisazione, entriamo subito nel vivo dell'argomento di questa

lezione, cioè nella descrizione particolareggiata della CPU.

Finora abbiamo sempre accennato all'unità centrale come a una sorta di "scatola magica", alla quale rivolgevamo in ingresso delle domande (seguendo una determinata sintassi e grammatica) e da cui ottenevamo in uscita delle risposte. Adesso è giunto il momento di rimuovere questa limitazione, cercando di scavare un po' più in profondità. La conoscenza - più o meno approfondita - dell'unità centrale non è solitamente richiesta al programmatore; anzi, nei limiti del possibile, tutti i linguaggi ad alto livello (come il BASIC) cercano di evitare che tra hardware e software esistano legami di interdipendenza.

La fase di scrittura di un certo programma (per lo meno in un linguaggio ad alto livello) dovrebbe infatti essere sempre svolta in assoluta autonomia dalle caratteristiche hardware del computer utilizzato, in modo da risentire il meno possibile delle peculiarità costruttive

# HARDWARE

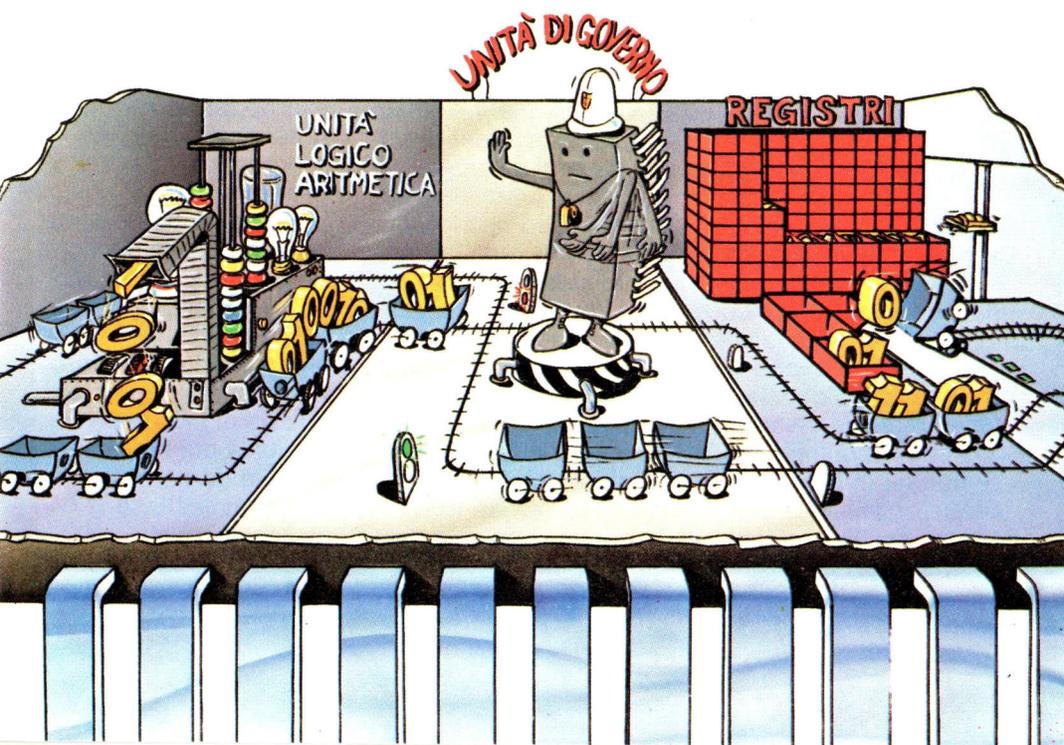
tipiche di un certo sistema.

Il BASIC, come ben sai - e soprattutto a causa di vari motivi storici che ne hanno provocato un'evoluzione anomala -, non può certo essere considerato il linguaggio in assoluto più "trasportabile" e "compatibile" (basta dare un'occhiata a uno stesso programma scritto per due diversi calcolatori per rendersene conto

immediatamente!). Negli ultimi anni sono stati tuttavia sviluppati numerosi altri linguaggi ad alto livello, che permettono di raggiungere l'obiettivo della portabilità (cioè della quasi perfetta compatibilità tra una macchina e l'altra) in modo praticamente completo.

Tuttavia, in determinate circostanze, soprattutto nei casi di utilizzo particolarmente "evoluto" del computer o di programmazioni in linguaggi a basso livello (cioè molto più vicini alla

macchina che all'uomo), può essere invece di estrema utilità fare specifico riferimento a un certo elaboratore e - di conseguenza - anche a uno specifico microprocessore. Oltre che a livello di cultura generale, la conoscenza della costituzione dell'unità centrale - per quanto semplificata e limitata - può pertanto risultare addirittura necessaria o addirittura indispensabile.



## Organizzazione esterna di una CPU

Un microprocessore da solo non sa fare niente. Perché possa lavorare occorre collegargli delle

memorie e delle interfacce. Una volta collegato a questi circuiti il microprocessore diventa allora un microcalcolatore, generalmente divisibile in quattro parti distinte, che comunicano le une con le altre tramite un bus dati, un bus indirizzi e un bus di controllo (per il momento puoi associare alla parola "bus" il corrispondente italiano di "collegamento"). Queste quattro parti sono:

- 1) Il microprocessore.
- 2) La memoria di programma, che segnala alla CPU le funzioni da compiere. Può essere - a seconda dei casi - memoria ROM o memoria RAM.
- 3) La memoria dati, che immagazzina i dati provenienti dalle periferiche, i risultati intermedi e i risultati finali.
- 4) Le interfacce, che consentono il trasferimento dei dati dalle periferiche verso il microprocessore e viceversa.

Esaminiamo per un momento più da vicino le ultime tre.

— La memoria di programma contiene la sequenza di istruzioni che il microprocessore

deve eseguire. Come già accennato, essa può essere sia di tipo RAM che ROM. Quando per esempio accendi il tuo computer la scritta che compare sullo schermo viene comandata alla CPU da un programma che si trova su ROM, mentre quando scrivi un'istruzione mediante tastiera essa viene memorizzata nella RAM.

— La memoria dati è invece un tipo di memoria notevolmente diverso dalla memoria del programma. È per forza di cose una memoria RAM, visto che deve poter essere letta o scritta tutte le volte che lo desideri. La sua presenza è indispensabile, poiché il microprocessore - per poter eseguire il suo programma - deve utilizzare dei dati che, per forza di cose, sono contenuti nella memoria dati o che provengono dalle periferiche tramite interfacce.

— Le interfacce infine, come ben sai, sono dei circuiti di ingresso-uscita la cui funzione è definita da un programma. Esse consentono la comunicazione tra il microprocessore e le periferiche.

## Organizzazione interna di una CPU

Un microprocessore contiene parecchie migliaia di transistor e di altri componenti elettronici: non è quindi il caso di descriverlo nei minimi particolari (e d'altra parte non ci sarebbe nemmeno di grande utilità).

Un microprocessore è comunque costituito da tre parti principali:

- l'unità di controllo
- l'unità aritmetico-logica

— i registri.

L'unità di controllo decodifica le istruzioni che vengono prelevate dalla memoria di programma ed elabora i segnali di comando necessari all'esecuzione di un'istruzione.

La funzione dell'unità aritmetico-logica (brevemente chiamata anche ALU) è invece l'esecuzione delle operazioni logiche ed aritmetiche sui dati che la alimentano attraverso le sue due porte di ingresso, che sono, rispettivamente, "l'ingresso destro" e "l'ingresso sinistro": queste porte si possono infatti immaginare come le due estremità più alte di una sagoma a forma di "V".

Dopo l'esecuzione di un'operazione aritmetica, come un'addizione o una sottrazione, l'ALU fa uscire i suoi contenuti dalla punta della "V".

I registri sono di due tipi: alcuni sono accessibili dal programma, altri sono interni alla CPU e non hanno un'importanza concettuale rilevante. I registri accessibili si dividono in tre categorie:

- i registri dei dati
- i registri degli indirizzi
- il contatore di

programma (Program Counter), il puntatore dello stack e il registro di stato.

I registri dei dati sono in pratica delle locazioni di memoria che consentono la memorizzazione temporanea delle informazioni negli spostamenti tra l'unità aritmetico-logica, le memorie e le interfacce. La loro lunghezza, cioè il numero di bit che compongono ciascun registro, è ovviamente uguale a quella della parola del microprocessore: 8 bit, se il microprocessore opera su 8 bit (come per esempio è il caso dello Z80 cioè della CPU montata nello Spectrum). I registri di indirizzo, detti anche puntatori, contengono indirizzi delle posizioni di memoria i quali vengono inviati al bus di indirizzo con un'istruzione particolare che consenta l'accesso a tali posizioni. Ti ricordi quando - parlando delle memorie - accennammo al fatto che ogni locazione disponeva di un ben preciso indirizzo? Bene, questo indirizzo serve proprio ai registri di indirizzo per fare in modo che la CPU possa

# HARDWARE

riferirsi a qualsiasi posizione della memoria. Il puntatore dello stack è un particolare registro di indirizzo che punta a una particolare zona della memoria, chiamata "area di stack". Viene automaticamente decrementato dopo ogni trasferimento di una informazione in questa zona, e incrementato dopo ogni prelievo. Il

puntatore di stack, come abbiamo già avuto modo di accennare in una delle scorse lezioni, ha una funzione particolarmente importante nella chiamata e nel ritorno dai sottoprogrammi. Il Program Counter (o contatore delle istruzioni) sorveglia l'esecuzione dell'intero programma. Viene caricato inizialmente con l'indirizzo della prima istruzione del programma, e in seguito segnala al microprocessore gli indirizzi delle istruzioni che devono essere eseguite successivamente. Mentre il microprocessore legge nella memoria di programma una istruzione, e la esegue, il contatore prosegue fino all'indirizzo dell'istruzione successiva e via di seguito. Il microprocessore in genere esegue le istruzioni in ordine sequenziale, cioè una dopo l'altra. Però, nel caso di alcune istruzioni l'esecuzione sequenziale di programma può essere modificata: nel Program Counter viene allora caricato l'indirizzo

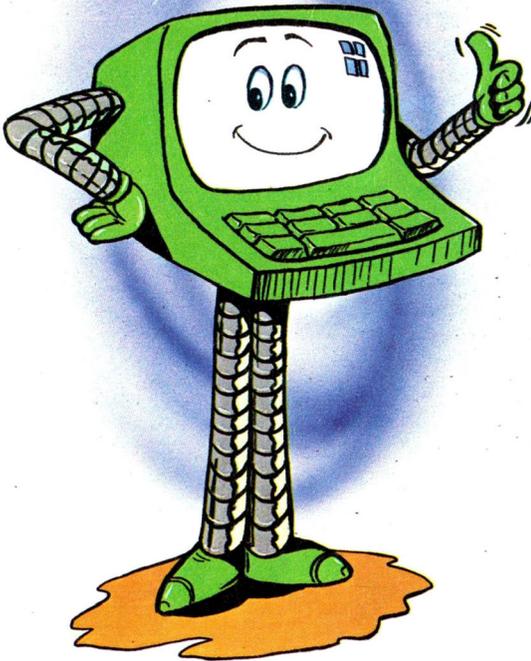
di salto, e l'eventuale indirizzo di ritorno al programma principale viene conservato per essere utilizzato in seguito.

Il registro di stato contiene invece un certo numero di bit posti a 0 o a 1, a seconda che si siano verificate determinate condizioni dopo l'esecuzione di alcune istruzioni (per esempio, se il risultato è positivo o negativo, con riporto o senza, ecc.). Lo svolgimento del programma può essere modificato in funzione dei valori assunti da uno o più bit del registro di stato. È uno dei registri più importanti che il programmatore ha a propria disposizione. I registri possono infatti essere usati per varie verifiche speciali o condizioni eccezionali, oppure per verificare velocemente alcuni risultati errati. Esiste infine un altro importantissimo registro che equipaggia la ALU: l'accumulatore. Questo è sempre uno dei due ingressi dell'ALU (poco importa se il "destro" o il "sinistro"); l'ALU fa infatti sempre automaticamente riferimento a questo accumulatore come ad uno degli ingressi.

# HARDWARE

Nelle operazioni aritmetiche e logiche, quindi, uno degli operandi sarà nell'accumulatore e l'altro si troverà tipicamente in una locazione della memoria. Il risultato sarà depositato nell'accumulatore. Il riferimento

dell'accumulatore come sorgente e destinazione dei dati è la ragione del suo nome: esso accumula i risultati. Il vantaggio di questo approccio basato sull'accumulatore è costituito dal fatto che si possono impiegare istruzioni molto più corte in linguaggio macchina. Se anche l'altro degli operandi dovesse infatti essere prelevato da uno degli altri registri (diversi dall'accumulatore), sarebbe necessario utilizzare istruzioni più lunghe per indicare l'indirizzo del registro. Perciò l'architettura dell'accumulatore si risolve in una maggiore velocità di esecuzione. Lo svantaggio è che l'accumulatore deve sempre essere caricato con i dati richiesti dall'operazione prima della sua utilizzazione. Questo può talvolta provocare qualche punto inefficiente.



# HARDWARE

## I bus

Il microprocessore comunica con le periferiche tramite tre bus:

- il bus degli indirizzi
- il bus dei dati
- il bus di controllo.

Il bus degli indirizzi è un insieme di linee elettriche, che consentono al microprocessore di selezionare una posizione di memoria o un registro di un'interfaccia. La CPU invia su queste linee, verso una periferica, un indirizzo codificato in binario. La periferica, ricevuto e decodificato l'indirizzo, seleziona il registro corrispondente. Il numero di linee del bus degli indirizzi determina quella che si chiama potenza di indirizzamento del microprocessore; per esempio, 16 linee consentono di indirizzare 2<sup>16</sup> (65536) locazioni di memoria. È da notare - tra l'altro - che il concetto di indirizzo è molto vicino a quello che si usa nel linguaggio corrente: la localizzazione di una persona in una città avviene in effetti tramite un indirizzo che contiene la strada e il numero civico della strada.

Il bus dei dati è anch'esso costituito da un gruppo di linee elettriche sulle quali avvengono gli scambi di dati tra il microprocessore e le periferiche (memoria ed interfacce). Il numero di linee di questo bus dipende dalla lunghezza della parola del

# HARDWARE

microprocessore: poiché il microprocessore dello Spectrum è a 8 bit, il bus dati dispone anch'esso di 8 linee.

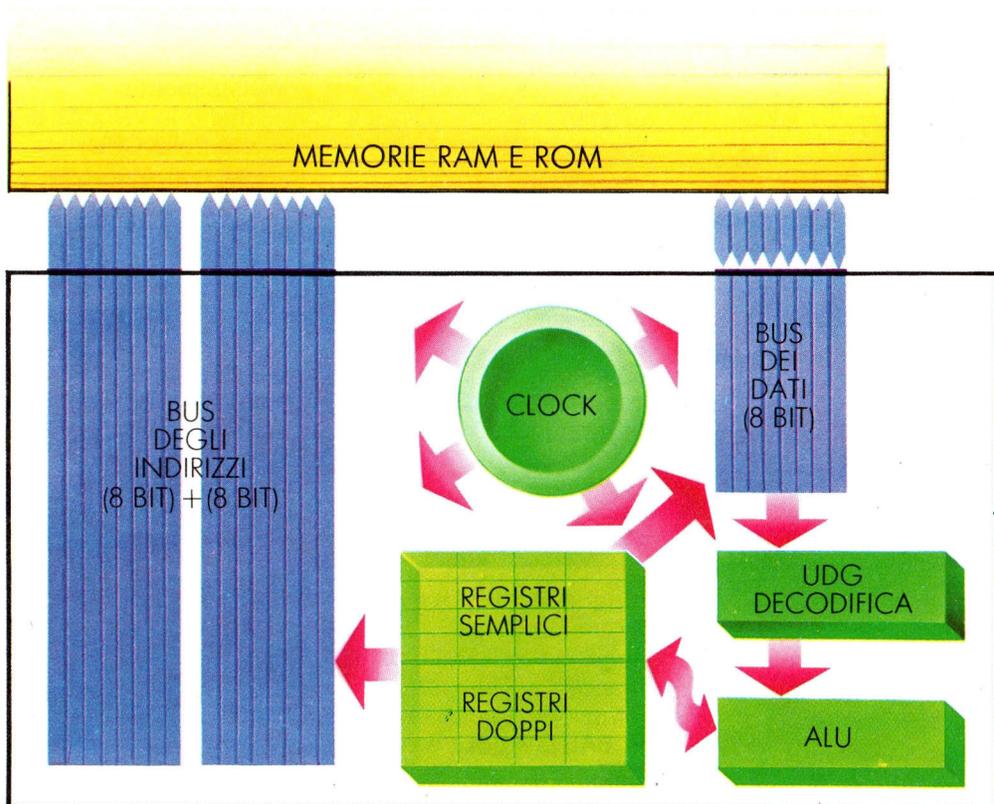
Il bus di controllo è costituito da un certo numero di segnali di vario genere, che assicurano la sincronizzazione tra il microprocessore e le periferiche. Le funzioni

più comuni assicurate da questo bus sono:

- la selezione di un'operazione di lettura o scrittura
- la interruzione del microprocessore (per interruzione si intende una procedura che - attraverso un segnale elettrico a un particolare piedino della CPU - segnala al

microprocessore che una periferica vuole comunicare con lui)

- la richiesta di accesso al bus di una periferica
- il riconoscimento di una richiesta di accesso al bus
- altre funzioni meno comuni, ma altrettanto importanti di quelle viste finora.



# LINGUAGGIO

## Il linguaggio macchina

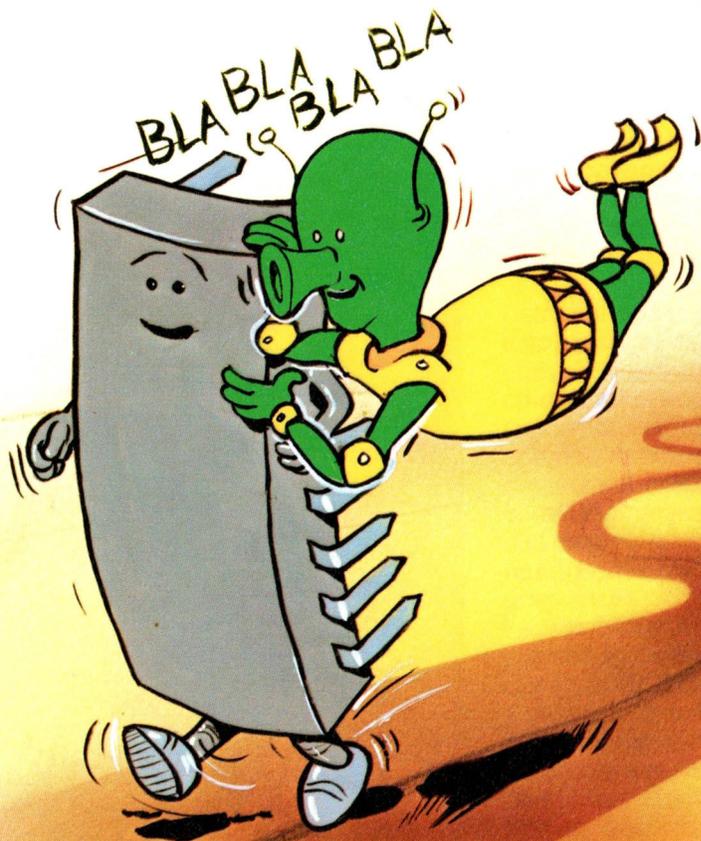
Abbiamo già detto varie volte che scrivere un programma significa scrivere una certa serie di istruzioni in un linguaggio comprensibile dal calcolatore, cioè in un linguaggio di programmazione. Facciamo per un momento un'analogia con l'uomo. Noi possiamo parlare molte lingue (inglese, francese, tedesco, ecc.), possiamo comprenderle tutte, ma se ci mettiamo a fare un ragionamento o a pensare a un problema, ci accorgiamo che ne usiamo una sola. Questa lingua, usata dal nostro cervello, a volte è la nostra lingua madre - l'italiano -, ma altre volte non è che una "lingua interna". Qualcosa di molto simile accade anche per i calcolatori. Essi possono infatti capire molti linguaggi di programmazione, il BASIC, il FORTRAN, il COBOL, il PASCAL, ecc., ma nel loro interno ne usano uno ed uno soltanto, formato da lunghe file di zeri ed uno in codice binario. Questo linguaggio è quello della macchina. Qualunque istruzione - per esempio in BASIC - deve essere tradotta in linguaggio macchina per poter

essere eseguita dal calcolatore. Tale compito nel BASIC del tuo Spectrum viene realizzato - ormai lo sai benissimo - dall'infaticabile interprete BASIC, vero e proprio traduttore simultaneo tra te e il tuo computer. In alcuni casi può tuttavia essere utile (o addirittura indispensabile) ricorrere alla programmazione diretta del microprocessore, scavalcando i normali meccanismi (che inevitabilmente rallentano la velocità di calcolo della CPU) imposti dall'interprete. Programmare in linguaggio macchina significa fornire al computer un certo insieme di configurazioni binarie (chiamate anche codici operativi) che il calcolatore è in grado di capire ed eseguire; ciascuna di esse rappresenta un'operazione eseguita via hardware dalla CPU, cioè mediante delle vere e proprie commutazioni di interruttori all'interno del computer. La differenza fondamentale è che per programmare in BASIC non è necessario conoscere come

# LINGUAGGIO

funziona il microprocessore con il quale è realizzato il calcolatore, mentre è necessario per

programmare in Assembler (così viene normalmente chiamato il linguaggio macchina). In BASIC, a meno di



# LINGUAGGIO

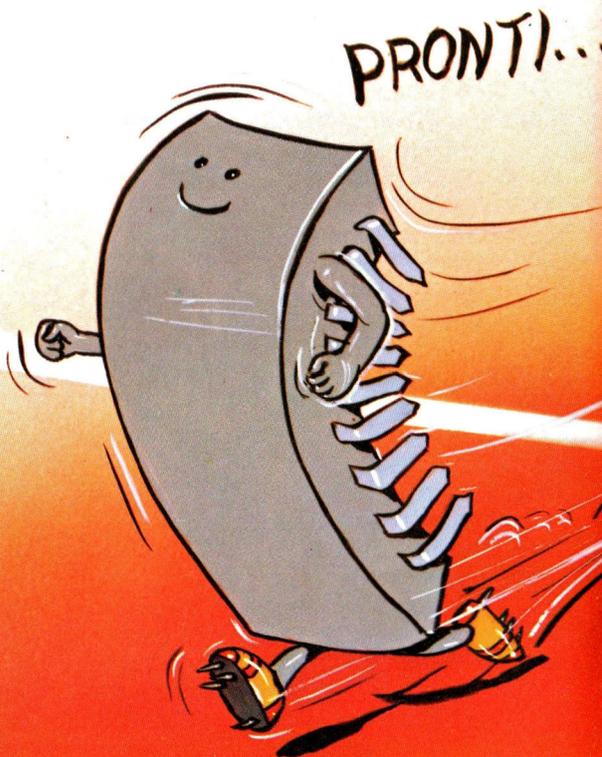
usare le istruzioni che leggono e scrivono direttamente nei byte di memoria (cioè PEEK e POKE) non devi mai occuparti di indirizzi di memoria: questo compito è affidato alle routine del sistema operativo e dell'interprete BASIC e viene effettuato in modo completamente automatico.

Una volta che hai imparato il linguaggio BASIC, esso ti mette a disposizione una specie di interfaccia software con la quale tu comunichi e che ti fornisce tutti i mezzi per programmare ed eseguire con successo i tuoi programmi.

I dati sono trattati con nomi simbolici (è cioè possibile assegnare un numero ad ogni nome: per esempio LET PREZZO = 5000); i riferimenti a punti specifici del programma (GOTO, GOSUB) si

ottengono con riferimento ai numeri distintivi delle linee del programma BASIC. In Assembler, invece, devi occuparti dei registri funzionali del microprocessore, degli indirizzi di memoria e

ogni operazione deve essere pensata in tutti i suoi particolari, seguendo le caratteristiche del calcolatore. Inoltre, in Assembler risulta meno semplice il trattamento dei dati.



# LINGUAGGIO

## Svantaggi e vantaggi dell'Assembler

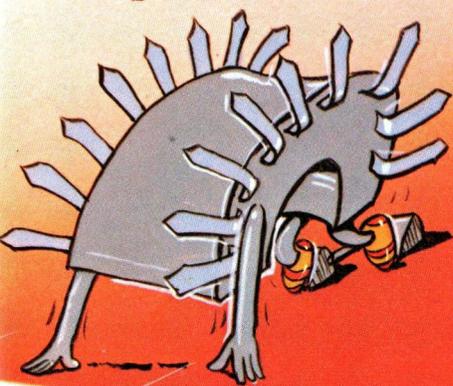
Il linguaggio macchina è in binario: puoi intuire come scrivere un programma in questo linguaggio sia lungo, laborioso e difficile.

D'altra parte i programmi in Assembler consentono una maggiore velocità operativa e un controllo sulla macchina molto più diretto che in BASIC, ed in certi casi è quindi

necessario o conveniente sobbarcarsi a questa fatica, quando l'applicazione lo richiede.

Esistono diverse possibilità per scrivere programmi in linguaggio macchina, per esempio utilizzando - anziché un codice numerico - un codice simbolico, in cui ciascuna istruzione viene rappresentata da una parola che in qualche modo ricorda l'operazione che l'istruzione stessa esegue. Ad esempio, l'istruzione di somma si esprime con ADD. Un codice di questo tipo si chiama mnemonico ("mnemonico" significa abbreviazione di una certa istruzione, che per la CPU corrisponde ad una determinata operazione); per essere proprio precisi l'Assembler è il linguaggio costituito da questi codici. Dato che il linguaggio comprensibile direttamente dal calcolatore resta comunque quello binario, occorre disporre di un programma che effettui la traduzione dall'Assembler al linguaggio macchina vero e proprio (cioè sostituisca al codice mnemonico di una certa

**VIA!**

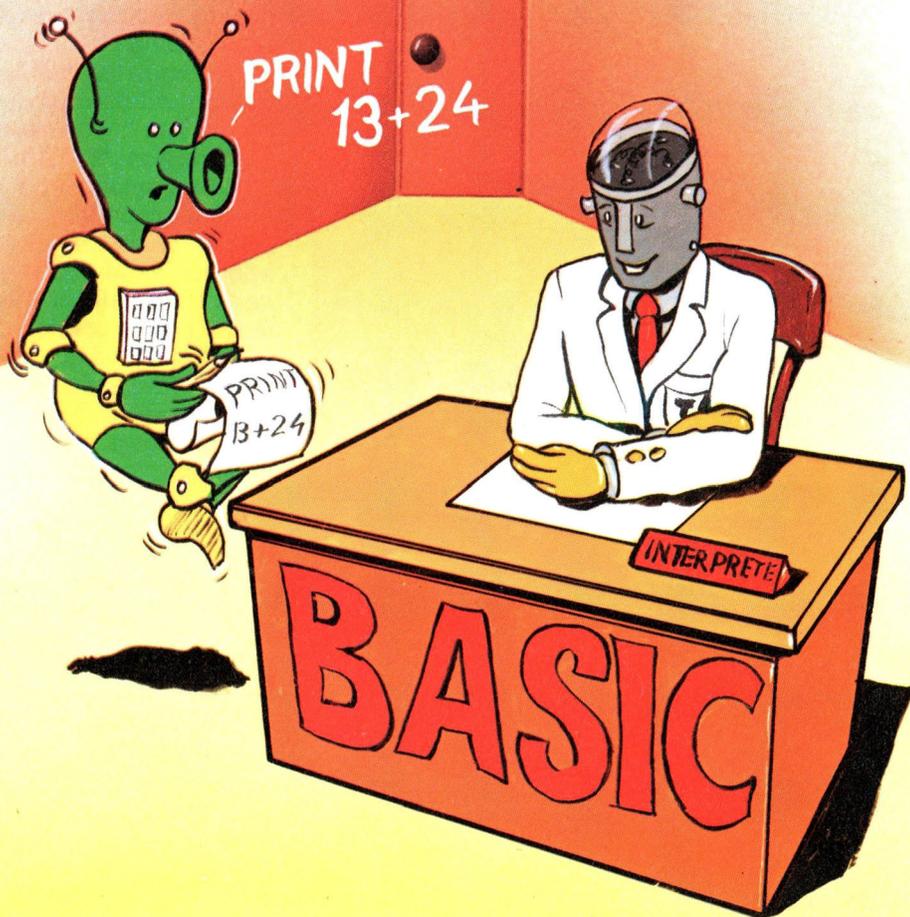


# LINGUAGGIO

istruzione il corrispondente codice numerico); questo programma prende il nome di assembler. Esistono in commercio numerosi assembler: il loro utilizzo, per chi

desidera programmare seriamente in Assembler, è praticamente indispensabile. Tenuto conto del carattere introduttivo al linguaggio macchina di questa e delle prossime lezioni,

non è comunque assolutamente necessario che ti premuri di procurartene uno. Per concludere, queste sono le principali cause che possono giustificare

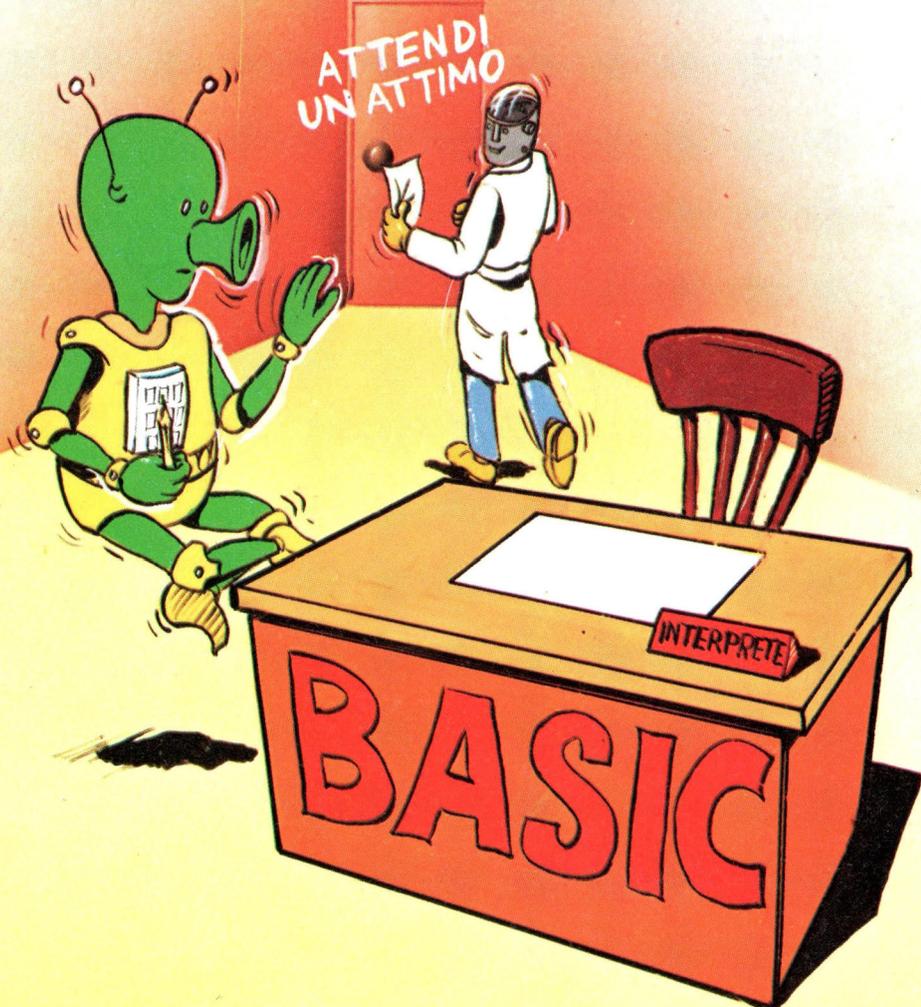


# LINGUAGGIO

l'impiego dell'Assembler:  
— tutte le volte in cui il  
fattore tempo gioca un  
ruolo fondamentale. Tra  
il tempo di esecuzione di  
un programma che  
funziona sotto  
l'interprete BASIC e

quello di un programma  
scritto direttamente in  
Assembler può esserci  
un'enorme differenza di  
velocità (il linguaggio  
macchina può essere  
fino a 2-300 volte più  
veloce);

— tutte le volte in cui il  
fattore spazio è di  
essenziale importanza.  
La dimensione di un  
programma BASIC,  
aggiunta a quella del  
suo interprete, sarà  
sempre superiore a



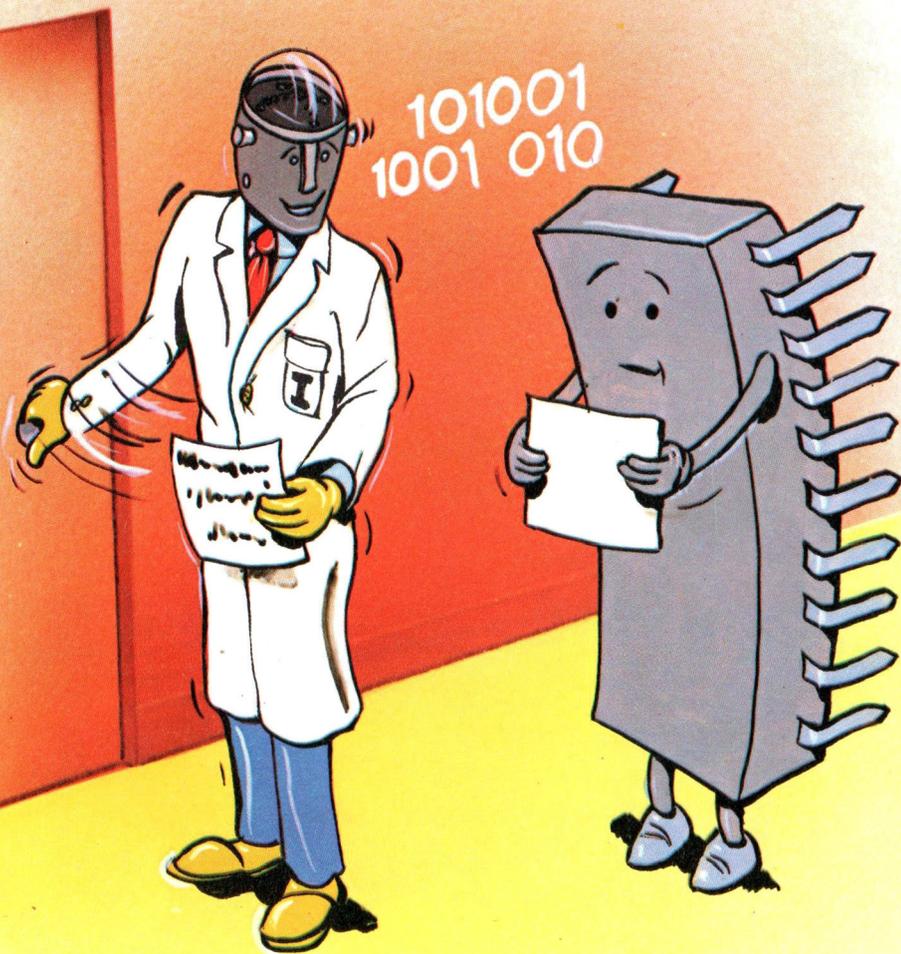
# LINGUAGGIO

quella dello stesso programma (che esegue la stessa funzione), realizzato in Assembler. Tale fattore, con il

progressivo ed inarrestabile calare dei prezzi delle memorie, sta comunque diventando sempre meno importante;

— tutte le volte in cui è necessario realizzare delle istruzioni sconosciute all'interprete BASIC, o impossibili da realizzare con un

linguaggio ad alto livello (come appunto il BASIC). Dato che un programma in Assembler risulta molto più veloce di un programma BASIC, ma ci vuole molto più tempo a scriverlo e risulta più difficile trovarne gli errori, sarà sempre necessario prendere di volta in volta una

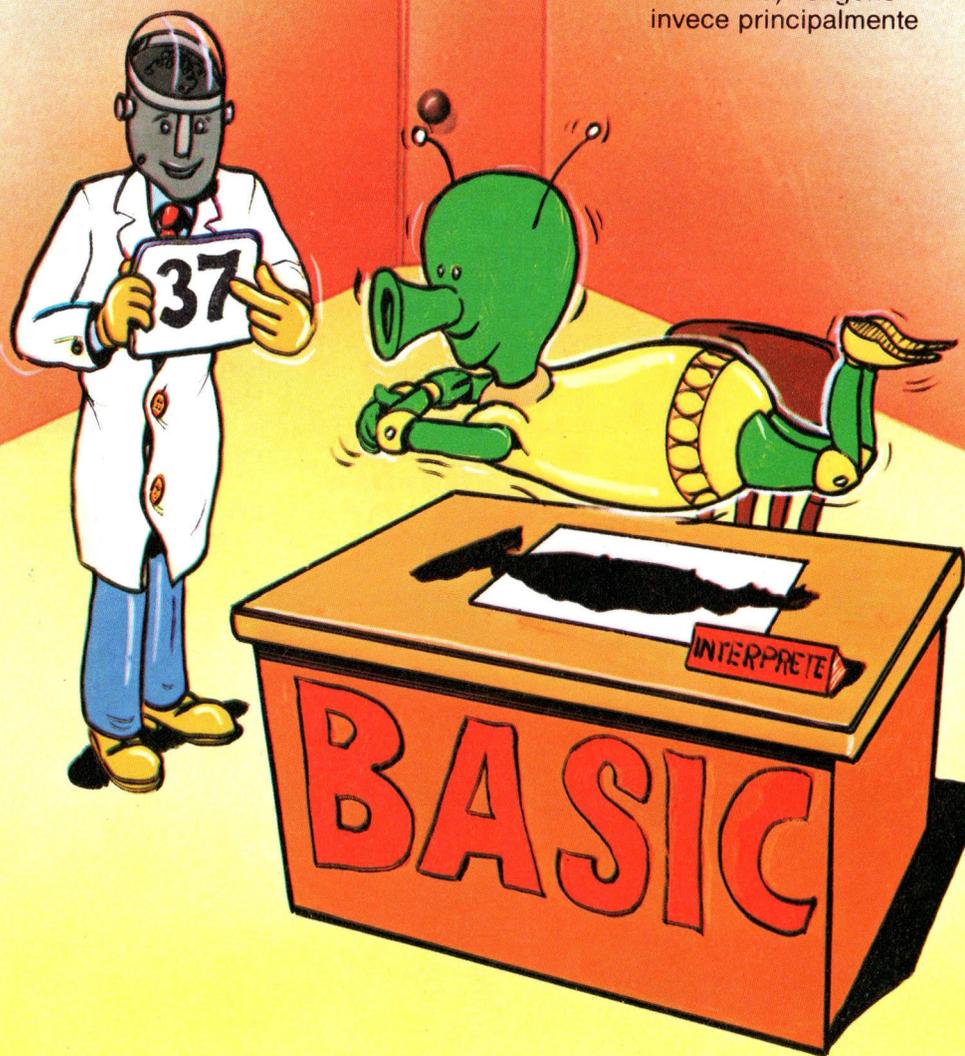


# LINGUAGGIO

decisione sulla strada che conviene scegliere, valutando bene le diverse esigenze, le possibili alternative ed i

rispettivi costi (oltre che in termini di tempo, anche di fatica). Inoltre, devi tener presente che certe particolari

applicazioni, come l'utilizzo del computer per giochi di animazione, si possono realizzare bene solo programmando in Assembler. In altri casi (esempio tipico: i programmi di gestione aziendale o di contabilità) vengono invece principalmente



richieste doti di leggibilità e di modificabilità dei programmi, ottenibili esclusivamente utilizzando un linguaggio ad alto livello.

Naturalmente, è sempre possibile (e lo si fa spesso) conciliare le due forme di linguaggio, scrivendo ad esempio l'ossatura del programma in BASIC e ricorrendo, quando è necessario, a dei piccoli programmi realizzati in Assembler. Esamineremo tra poco questa possibilità.

---

## USR

---

USR è un comando del BASIC che significa "User SubRoutine" (routine definita dall'utente). Esso risulta molto utile quando si desidera passare direttamente l'esecuzione da un programma BASIC a una routine scritta in linguaggio macchina. Naturalmente, occorre che al momento della chiamata di USR il programma in linguaggio macchina sia già stato memorizzato all'interno dello Spectrum. È inoltre necessario - proprio come avviene nella chiamata delle subroutine in BASIC - specificare l'indirizzo di partenza della routine che si desidera eseguire. Per essere più precisi USR è quindi una funzione: essa richiede infatti un argomento ben definito, corrispondente alla locazione di memoria da cui dovrà cominciare l'esecuzione del programma in linguaggio macchina, e fornisce un risultato secondo le regole che vedremo più avanti. Così, se scriviamo un programma contenente

questa linea BASIC:

100 PRINT USR 32000

il nostro Spectrum eseguirà nell'ordine questa serie di operazioni:

- 1) Passerà il controllo dal BASIC alla routine in linguaggio macchina, che in precedenza sarà stata memorizzata a partire dalla locazione 32000.
- 2) Eseguirà e porterà a termine (naturalmente quando questa eventualità sarà stata prevista dal programmatore) le varie istruzioni in linguaggio macchina.
- 3) Ritournerà il controllo della macchina all'interprete BASIC, che provvederà a stampare sullo schermo - prima della prosecuzione alle linee successive - il contenuto (cioè il valore numerico) di una coppia di registri (precisamente il B e il C) del microprocessore. Chiariamo meglio quest'ultimo punto. Al termine del sottoprogramma in linguaggio macchina il sistema operativo dello Spectrum riprende in mano la situazione. La

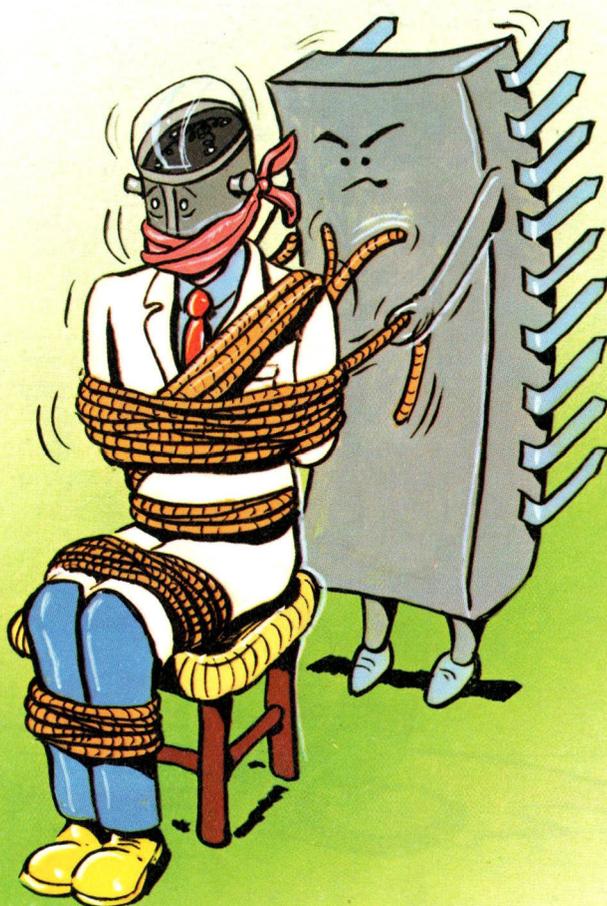
# LINGUAGGIO

funzione USR, tra l'altro, fa sì che il valore da essa assunto al ritorno dalla routine in

linguaggio macchina (ricordati che le funzioni in uscita forniscono sempre un risultato) venga posto uguale al valore contenuto nella coppia di registri BC dello Z80 (cioè della CPU dello Spectrum). Il microprocessore dispone infatti di numerosi registri -

utilizzati per effettuare i vari calcoli - convenzionalmente chiamati con alcune lettere dell'alfabeto. Tra questi vi sono anche il B e il C. Se avessimo scritto:

100 LET A = USR32000



# LINGUAGGIO



al termine della routine  
in linguaggio macchina il  
valore contenuto in BC

sarebbe stato invece  
memorizzato nella  
variabile A.

In alcune circostanze  
però può non essere di  
alcuna utilità pratica  
visualizzare sullo  
schermo o memorizzare  
in una variabile  
numerica il contenuto  
della coppia BC. In tal  
caso si può fare uso di  
un

RANDOMIZE USR 32000



(sempre ammettendo di aver memorizzato la routine in linguaggio macchina a partire dalla locazione 32000). Ricorrendo a questa possibilità tutto procede

infatti in modo assolutamente identico a prima; l'unica differenza sta nel fatto che stavolta il risultato della funzione USR andrà ad alterare la variabile di controllo usata per generare i vari valori di RND. Questo fatto può essere utilmente sfruttato quando, per esempio, si desidera progettare un programma che muova degli oggetti sullo schermo in modo apparentemente casuale. Se invece la generazione dei numeri non è di alcun interesse, poco male: l'importante è che la routine in linguaggio macchina sia stata eseguita e che il ritorno al BASIC sia avvenuto senza "sporcare" lo schermo o occupare memoria. Per quanto gli esempi esaminati più avanti (nella parte della lezione dedicata alla programmazione)

chiariscano questo fatto, è importante ricordare che con USR il computer esegue una vera e propria chiamata a una subroutine (in questo caso scritta - lo ripetiamo - in linguaggio macchina). In altre parole, dopo aver fatto eseguire allo Spectrum una istruzione del tipo PRINT USR... o RANDOMIZE USR... è necessario inserire un'istruzione in codice macchina, che dica al computer di tornare al BASIC. È la stessa cosa che succede quando si usa un'istruzione GO SUB; per tornare al programma principale occorre inserire un comando RETURN al termine del sottoprogramma. Questa istruzione, nell'Assembler dello Z80, ha come codice mnemonico RET e come codice operativo il valore decimale 201.

## Sintassi della funzione

---

USR indirizzo

---

# PROGRAMMAZIONE

## Alla scoperta dell'Assembler

Prima di passare direttamente alla scrittura di programmi in linguaggio macchina, dobbiamo ancora affrontare importanti (anzi, indispensabili) argomenti, grazie ai quali potremo successivamente

avanzare nei nostri discorsi in modo più semplice e spedito. Ti potrà forse sembrare che il linguaggio macchina non faccia per te: troppe cose da imparare, da sapere e da tenere a mente. Questo può anche essere vero: tuttavia una volta imparati i concetti fondamentali, molti argomenti ti diventeranno molto meno complicati di quello che potevano apparire a prima vista. Inoltre più ti addenterai nell'assembler - diventando di conseguenza più padrone della situazione - più i risultati diventeranno spettacolari e carichi di soddisfazioni.

## La numerazione esadecimale

Il primo, importante punto è determinare un sistema di numerazione che giunga a un compromesso tra la difficoltà che l'uomo incontra nel leggere e lavorare con i numeri binari e l'avversità che la CPU possiede verso le cifre decimali. Ecco quindi che interviene un terzo e

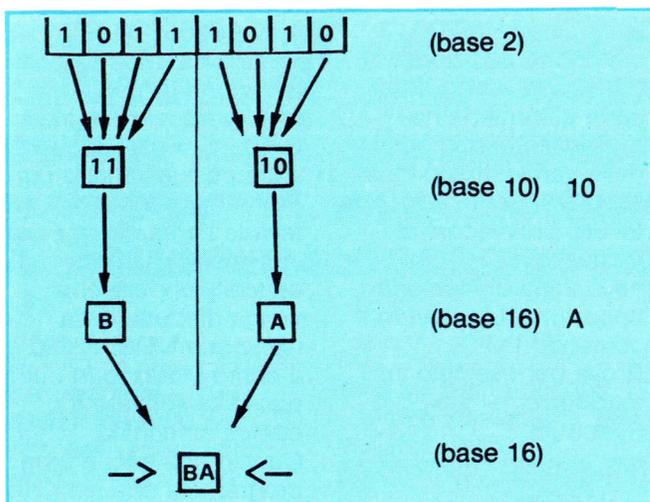
fondamentale (per chi programma in Assembler) sistema di numerazione, cioè quello in base 16. Con questa base un numero binario a otto cifre può essere scritto con due cifre in base sedici, cioè con due numeri esadecimali. Ricorderai certamente che un numero in base due è composto solo da cifre 0 e 1, mentre uno in base 10 è scritto con cifre comprese tra 0 e 9; così un numero in base 16 dovrà essere scritto con cifre comprese tra 0 e 15. Però, dal momento che non abbiamo cifre maggiori di 9, usiamo le prime sei lettere dell'alfabeto:

10 = A  
11 = B  
12 = C  
13 = D  
14 = E  
15 = F

Il modo più semplice di convertire un numero binario a 8 bit in un numero esadecimale è quello di separare gli 8 bit in due gruppi di 4 bit, effettuare la conversione da una base all'altra di ciascun gruppo e poi ricombinare il risultato.

# PROGRAMMAZIONE

Per esempio:



Con questo metodo possiamo convertire qualunque valore di un byte in due cifre esadecimali. Il vantaggio è quello di avere una scrittura più compatta della notazione binaria e, con un minimo di allenamento, quasi identica al sistema decimale. In ogni caso, tutti questi sistemi di numerazione sono - ricordiamolo - modi diversi di rappresentare gli stessi numeri. Tutto sta nel conoscere la base utilizzata. Per convenzione, allo scopo di riconoscere immediatamente un numero esadecimale,

scriveremo sempre i numeri in base 16 con il suffisso H. Così

10

significherà "dieci decimale", mentre

10H

significherà "uno zero esadecimale" (cioè 16 decimale).

## Come memorizzare i programmi in linguaggio macchina

Puoi usare diversi metodi per memorizzare i programmi in linguaggio macchina. Il più semplice è quello di abbassare il TOP della memoria dedicata al programma BASIC, agendo sul puntatore RAMTOP. Se ti servono i caratteri grafici utente basterà abbassare il contenuto di RAMTOP, in modo tale da non invadere la zona dei caratteri grafici utente.

# PROGRAMMAZIONE

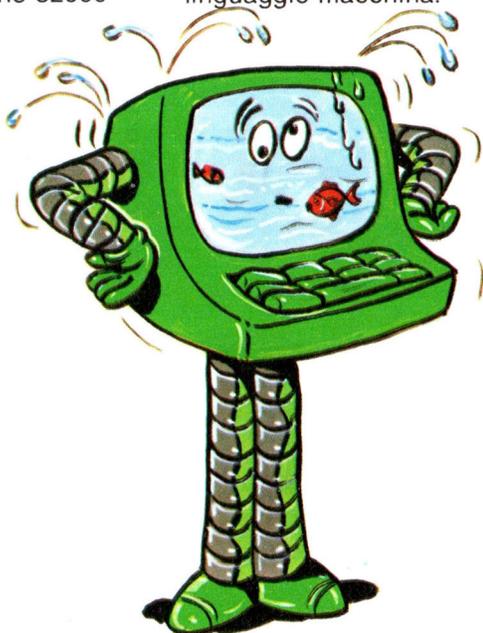
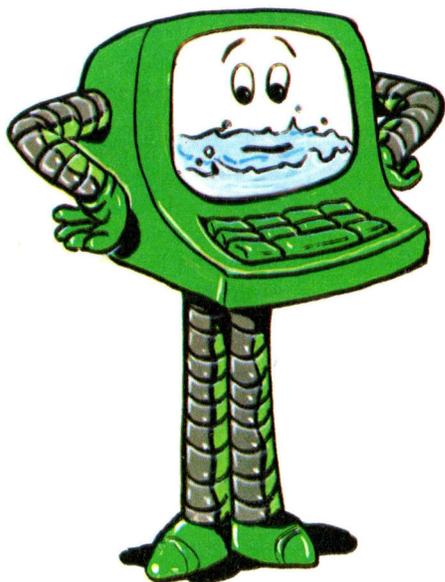
L'abbassamento del TOP della memoria si ottiene con il comando CLEAR seguito dall'indirizzo dell'ultimo byte che vuoi avere disponibile per il programma BASIC. Le routine in linguaggio

macchina possono essere incorporate nei programmi BASIC, usando le istruzioni DATA seguite da una serie di numeri che rappresentano i codici delle varie istruzioni da eseguire. Poi una routine BASIC provvederà a trasferire i valori dei byte nella zona di memoria opportuna, utilizzando il comando POKE. Prova per esempio ad introdurre nel tuo Spectrum:

```
CLEAR 32000
```

Cancellerai la memoria dalla locazione 32000

alla fine della RAM (32767 nella versione a 16K e 65535 nella versione a 48K). Questa istruzione proteggerà una eventuale routine in linguaggio macchina dalla possibilità che un programma BASIC invada l'area di memoria in cui è scritta e la cancelli, oppure che venga distrutta dalla istruzione NEW. 32000 è il primo indirizzo in cui è possibile scrivere il codice macchina. Questo che segue è un programma che ti permetterà di caricare, una dopo l'altra, le istruzioni delle routine in linguaggio macchina:



# PROGRAMMAZIONE

```
10 INPUT "RAMTOP ="; RT
20 CLEAR RT
25 RESTORE
30 LET INIZIO = 1 + (PEEK 23730 + 256 * PEEK
  23732)
40 FOR A = INIZIO TO 65535 (usa 32767 nella
  versione 16K)
50 READ X : IF X = 999 THEN STOP
60 POKE A, X
70 NEXT A
80 DATA ... qui vanno scritti i codici ...
```

Esaminiamo il programma linea per linea, per vedere come funziona: la linea 10 chiede l'indirizzo di RAMTOP. La linea 20 fissa la RAMTOP alla locazione indicata. La 25 fa uso dell'istruzione RESTORE per posizionare il puntatore

data sul primo dei dati. La linea 30 assegna alla variabile INIZIO la locazione di partenza del programma in LM. Non si può fare:

```
INIZIO = RT + 1
```

perché l'istruzione CLEAR cancella anche tutte le variabili: poiché le locazioni 23730 e 23731 contengono l'indirizzo dell'ultimo byte disponibile per il BASIC, basterà leggerne il contenuto per stabilire il corretto valore di INIZIO. La linea 40 inizia un ciclo che va dalla locazione INIZIO al massimo indirizzo possibile sullo Spectrum (65535 nella versione a 48K e 32767 in quella a 16K). La linea 50 legge i vari elementi delle righe DATA e controlla man

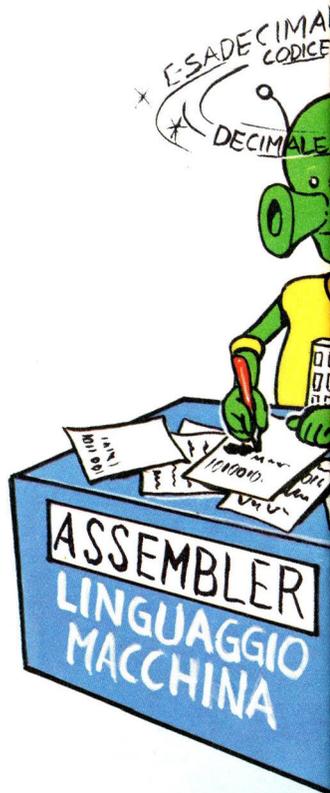
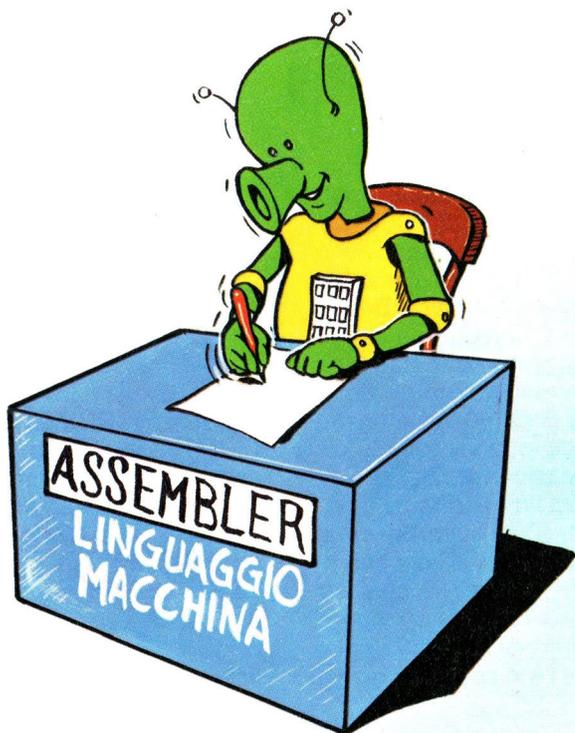
mano se ve n'è uno pari a 999 (che è un codice non ammesso per l'istruzione POKE); in questo modo siamo in grado di identificare la fine dei dati. La linea 70 chiude il ciclo. Naturalmente, non è necessario avere tutti i valori su un'unica linea DATA; puoi usare quante linee desideri, compatibilmente con la memoria disponibile. La linea 80 (e quelle eventualmente seguenti) contiene i valori decimali della routine e deve avere come ultimo elemento il 999, affinché il computer possa accorgersi di essere arrivato in fondo. Così questo programma carica il codice macchina nella RAM. Per far eseguire la routine occorrerà modificare la linea 50, in modo tale che in corrispondenza del 999 il programma non si arresti, ma vada a comandare una linea PRINT USR INIZIO oppure RANDOMIZE USR INIZIO. Questo esercizio (tutto sommato veramente elementare) lo lasciamo fare a te. In alternativa puoi comandare l'esecuzione impartendo uno dei due comandi USR in modo immediato.

# PROGRAMMAZIONE

## Esempi Assembler

Proviamo adesso a scrivere qualche programma molto semplice in Assembler, illustrandone contemporaneamente il funzionamento e il confronto con il corrispondente listato BASIC. Non sempre è possibile fare tale paragone: gli esempi che tratteremo ci

daranno tuttavia questa possibilità. Come primo esempio proponiamoci di trasferire il contenuto di due locazioni di memoria (per esempio la 5428 e la 5429) in un'altra coppia di locazioni (scegliamo la 25000 e la 25001). In BASIC scriveremmo:



# PROGRAMMAZIONE

10 POKE 25000, PEEK 5428  
20 POKE 25001, PEEK 5429

e tutto sarebbe risolto.  
Vediamo adesso come  
fare in Assembler. Ecco  
qui il listato:

```
LD HL,(1534H)  
LD (61A8H), HL  
RET
```

La prima riga carica (LD  
è abbreviazione di  
LOAD, cioè carica) il  
registro HL con il

numero contenuto nella  
coppia di locazioni poste  
a partire dall'indirizzo  
5428 (1534  
esadecimale). La  
seconda memorizza  
nella coppia di locazioni  
61A8 e 61A9 (25000 e  
25001 decimali) il valore  
contenuto in HL. La terza  
riga è il famoso  
RETURN, che serve per  
restituire il controllo al  
BASIC.

È interessante osservare  
che abbiamo in pratica  
lavorato su 16 bit alla





# PROGRAMMAZIONE

numerici eseguibili (questi che abbiamo appena visto sono solo i mnemonici). Occorre prendere in mano la

tabella fornita dal costruttore del microprocessore, e convertire uno ad uno i vari termini. A ciascun elemento corrisponderà naturalmente un preciso codice numerico:

LD HL, (1534H)	= 2A 34 15
LD (61A8H), HL	= 22 A8 61
RET	= C9

I numeri che abbiamo scritto sono ancora in esadecimale: prima di essere scritti nelle DATA dovranno essere convertiti in decimali. L'ultima cosa da notare è che - se guardi bene - gli indirizzi delle locazioni sono stati scritti ribaltati (cioè 1534 è stato scritto 3415, e analogamente per 61A8). Ciò è dovuto a ragioni costruttive del microprocessore, il quale pretende che di qualsiasi indirizzo gli vengano forniti prima il byte basso e poi quello alto.

Alla fine otteniamo i seguenti valori:

2AH = 42
34H = 52
15H = 21
22H = 34
A8H = 168
61H = 97
C9H = 201

che, introdotti nella linea DATA (ricordandosi di aggiungere anche il 999), formeranno la routine.

Esegui il programma e comanda quindi unaUSR. Prova quindi a verificare con delle PEEK nelle locazioni 25000 e 25001 - che effettivamente il lavoro sia stato compiuto!

# PROGRAMMAZIONE

Proviamo ora questo secondo esempio:

```
LD B,00H ;carica 00H nel registro B
LD C,00H ;carica 00H nel registro C
LD A,0CH ;carica 0CH nel registro A
SUB A,06H ;sottrae 06H da A
LD C,A ;carica il valore di A in C
RET ;ritorna al BASIC.
```

Da notare che in Assembler i commenti vengono separati dalle istruzioni mediante un punto e virgola. L'equivalente BASIC del programma appena visto sarebbe:

```
10 LET B = 0
20 LET C = 0
30 LET A = 12
40 LET A = A - 6
50 LET C = A
60 RETURN
```

La conversione del programma nei codici numerici porta ai seguenti valori:

```
6, 0, 14, 0, 62, 12, 214, 6, 79, 201,
```

Se eseguirai questa routine mediante una PRINT USR INIZIO, vedrai che sullo schermo apparirà - come abbiamo già detto in precedenza - il contenuto dei registri B e C. Essi varranno rispettivamente 0 e 6.

## Conversione di un carattere in LM

Al di là della reale possibilità di utilizzo, l'obiettivo importante di questa proposta è quello di sottolineare il corretto procedimento da seguire per impostare ed eseguire un qualsiasi programma in linguaggio macchina.

Per convertire, infatti, un carattere minuscolo nel corrispondente maiuscolo, puoi usare benissimo il BASIC, ma l'esempio ci dà l'opportunità di evidenziare le 3 fasi fondamentali necessarie ad introdurre e a utilizzare routine in LM nei tuoi programmi.

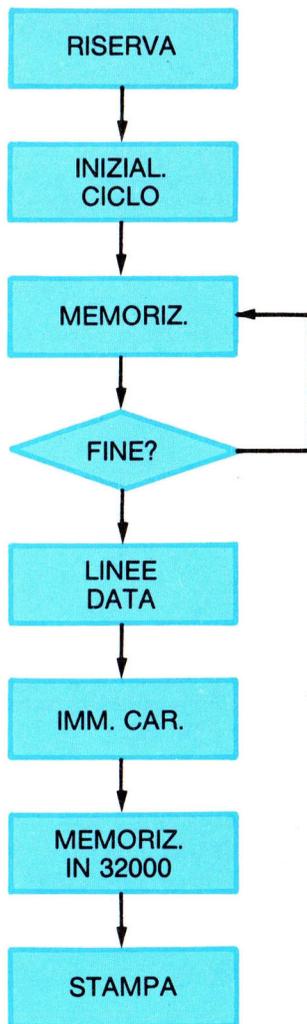
# PROGRAMMAZIONE

- 1 Preparazione:**  
Si fissa la RAMTOP affinché il programma BASIC non possa in alcun modo sovrapporsi al codice macchina.
- 2 Memorizzazione:**  
si introduce in memoria il codice macchina.
- 3 Esecuzione:**  
si lancia il programma in LM.

PREPARAZIONE

MEMORIZZAZIONE

ESECUZIONE

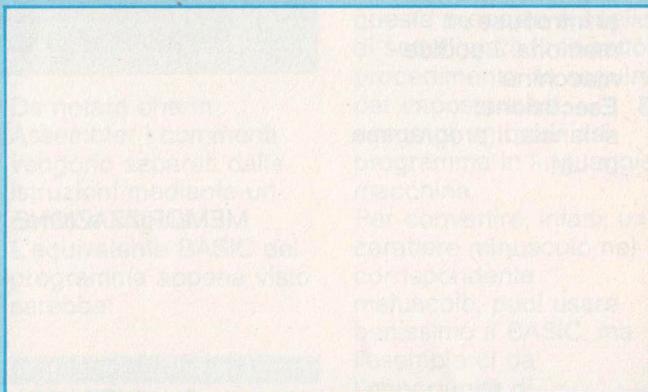


```
10 CLEAR 31999
20 FOR I = 32001 TO 32009
30 READ X
40 POKE I, X
50 NEXT I
60 DATA 58, 0, 125, 203, 175, 79, 6, 0, 201
70 INPUT C$
80 POKE 32000, CODE C$
90 PRINT CHR$ USR 32001
```

# VIDEOESERCIZI

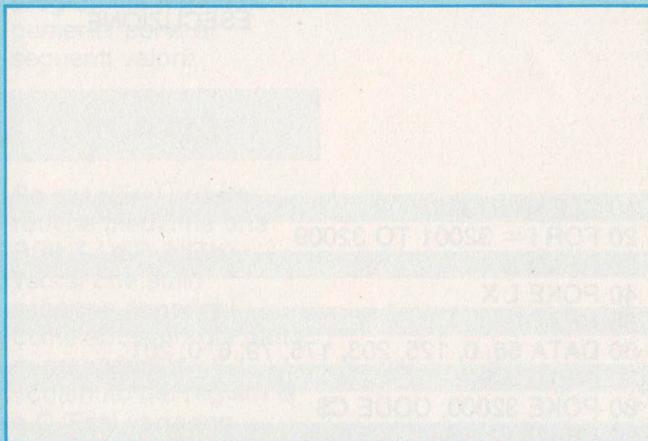
Cosa resterà sullo schermo al termine dell'esecuzione di questo programma?

```
10 PRINT AT 10, 13; FLASH 1; "PROVA"  
20 PRINT AT 11, 6; "DI UNA STRANA ROUTINE"  
30 PRINT AT 12, 6; "IN LINGUAGGIO MACCHINA"  
40 PRINT USR 0  
50 PRINT "SE PASSA DI QUI È UN MIRACOLO!"
```



Lancia il seguente programma e, tramite esso la routine del sistema operativo che inizia alla locazione 3582. Scoprine da te l'effetto variando il contatore del ciclo automatico.

```
10 LIST : LIST  
20 PAUSE 200 : BEEP 1, 20  
30 FOR V = 1 TO 7  
40 LET A = USR 3582  
50 NEXT V  
60 REM GUARDA IN SU
```







**GRUPPO  
EDITORIALE  
JACKSON**